The Bell System Technical Journal
Vol. 52, No. 10, December, 1973
Printed in U.S.A.

Information Management System:

The Natural Dialogue System

By B. W. PUERLING and J. T. ROBERTO

(Manuscript received October 5, 1972)

The Natural Dialogue System (NDS) is a software system designed to permit the easy implementation of time-shared computer programs which employ sophisticated forms of man-machine dialogue to converse with members of a nonprogrammer user audience. The heart of the system is a syntax-directed translator which recognizes user input messages and translates them into an internal text of integers for use by the program. NDS allows the language designer to specify the syntax of the statements in his language, the form of their translations, methods for diagnosing errors in user's input, diagnostic messages to be generated, and the style of dialogue which will exist between the programs and their users. This is accomplished through a dialogue description and a language description consisting of syntactic specification elements with semantic procedures embedded within them. Use of NDS allows the language designer to produce an interactive language which is tailor-made for both his users and his programs. NDS relieves the language designer of the necessity of writing a complex message analyzer, thereby substantially reducing the effort required to produce systems that offer these forms of man-machine dialogue. Furthermore, use of NDS allows such systems to be implemented by less sophisticated programming talent than would otherwise be necessary.

I. INTRODUCTION

The Natural Dialogue System (NDS) is a tool to aid programmers in the implementation of time-sharing-based computer systems which employ keyword-oriented languages and a variety of styles of man-machine dialogue to converse with members of a nonprogrammer user audience. By keyword-oriented languages we mean languages of the type illustrated by Sinowitz[1] and suggested as an alternative to natural

language for communicating with information management systems by Chai.[2] NDS provides the designer of such a language witb the ability to define tbe syntax of the statements in the language, tbe forms of tbeir translations, methods for detecting errors made by users of tbe language, and diagnostic messages to be sent to users when errors are detected. In addition, NDS provides facilities for tbe language designer to specify the style of dialogue which will exist between the system and its users.

NDS bas been operational on an experimental basis since 1970. It bas been implemented under five different host operating systems (including one batch system). Its primary use has been in the area of interactive query languages for information management systems, including inventory management systems, a budget control system,[3] a work force administration system, information retrieval systems bascd on surveys of financial and equipment data, and a general-purpose hierarchical data base management system.[4] Other uses have included a data checking specification language, a report generator composition language, and bulk data input/output format specification languages. NDS has also led to furtber work in the area of tools for interactive language design presented by Heindel and Roberto.[5]

In order to get a feel for what can be done using NDS, some concepts concerning styles of man-macbine dialogue are presented, followed by a description of the styles of dialogue obtainablo using NDS. NDS itself is tben described, including an overview of the modules of tbe system and certain details concerning the specification of systems to be implemented using it.

II. DIALOGUE CONCEPTS

In making conversational software systems available to non-programming audiences for purposes of information retrieval and problem solving in general, a broad spectrum of conversational styles has evolved. At tbe extremc ends of this spectrum we have machine-initiated dialogue and user-initiated dialogue. In the machine-initiated style, the user is asked questions by the computer. These questions are designed to find out, in an orderly way, what the user wishes tbe system to do for him. In the user-initiated style, the user presents his problem to the system and directs its action. The two styles are best illustrated by example.

(*i*) Machine-initiated dialogue.

Computer: WHAT IS THE VALUE OF EXPENSE?

User: 50000

Computer: DO YOU WANT THE VALUE OF PROFIT?
User: YES
Computer: PROFIT IS 40000

(*ii*) User-initiated dialogue.
User: EXPENSE IS 50000. WHAT IS PROFIT?
Computer: PROFIT IS 40000

Coupled with these different styles of dialogue is the problem of conversational dynamics where at some point in time during the conversation the subservient participant wishes to seize the initiative. For example:

(*i*) User seizes initiative from computer.
Computer: WHAT IS EXPENSE?
User: IGNORE EXPENSE. REVENUE IS 80000.

(*ii*) Computer seizes initiative from user.
User: TAX IS 5%. WHAT IS PROFIT?
Computer: PROFIT CANNOT BE COMPUTED YET, WHAT IS EXPENSE?
User: 50000

With either of these styles, a user is apt to input information which is syntactically or semantically incorrect. Input of this nature should not cause the conversational program to abort. On detecting invalid input a conversational program may output terse messages such as "WHAT?" or "SYNTAX ERROR" and then invite re-entry of the test in question. Alternatively, programs may output lengthy explanations of valid replies and again ask the user to continue his input. The nature of handling invalid input depends primarily on the experience level of the end-users as well as the experience level of the person implementing the conversational software. In general, a language designer should have the tools at his disposal to tailor his language, including handling of invalid input, to correspond precisely to the environment in which it will be used.

In general, a transaction between man and machine can be viewed as a consulting effort between two "experts":

(*i*) the machine, which is an expert in delivering facts or computing results based on input data, and

(*ii*) the person, who is an expert in the problem to be solved, the environment in which the problem arose, and certain subjective considerations of the possible solutions.

In this situation if the person is burdened by certain conversational

constraints, the creative, exploratory environment may suffer. In other words, the conversation between man and machine should be made as natural as possible for the person. With this in mind, NDS offers a variety of dialogue styles which encompass most of the initiative spectrum with the emphasis on the user-initiated end.

III. STYLE OF NDS DIALOGUE

A user's message or request to a system using NDS consists of a series of statements separated by colons. Each statement in the language consists of a unique keyword followed by a sequence of characters called the clause of the statement. A message is ended by the statement GO:. A message to an information management system might be:

PRINT ITEM NUMBER, SELLING PRICE/PURCHASE COST, REORDER DATE: WHEN AMOUNT IN STOCK > 1000: IN ALL DEPARTMENTS: GO:

This message consists of three statements with the keywords PRINT, WHEN, and IN respectively. The PRINT statement fills the same role as a verb in the English language since it directs the information system to print the information specified in its clause. In general, a user's message must contain a single verb statement. The WHEN and IN statements act as modifiers (adverbial or prepositional) of the PRINT verb. In general, a user's message contains zero, one, or more modifier statements. The statements in a message can be given in any order since NDS does not consider a message to be complete until the GO statement is encountered.

An important feature which NDS offers is the automatic edit mode. NDS remembers the state of the dialogue from message to message. Once a statement is correctly given by a user, that statement remains as part of the "current" message until the user deletes it, or replaces it. Therefore, after the system acts on the above request, the user may continue the dialogue by typing:

WHEN AMOUNT IN STOCK > 2000: GO:

The PRINT and IN statements which were given as part of the first message are carried over as part of the second message. Therefore, to NDS the second message becomes:

PRINT ITEM NUMBER, SELLING PRICE/PURCHASE COST, REORDER DATE: WHEN AMOUNT IN STOCK > 2000: IN ALL DEPARTMENTS: GO:

Once a verb statement is correctly given by a user, that statement remains as part of the current message until the user deletes it, replaces it, or enters the statement for a different verb in the language. Thus, following action on the second message, the user may continue his dialogue by typing:

DISTRIBUTE ITEMS BY SELLING PRICE: GO:

The IN statement from the first message and the WHEN statement given in the second message are carried over as part of the third message. If the user continues his dialogue by inputting a statement whose clause has an invalid construct according to the definition of the clause given by the language designer, the system will print a diagnostic message (possibly language designer defined) and remove the statement from the current state of the dialogue.

In addition to verbs and modifiers, a language may contain one or more special statements termed dialogue service statements. These statements usually take the form of aids to the user of a language or debugging tools for the language designer. Services may be included which provide the user with explanations of terms used in the language, news of recent changes to the application, instructions on the use of the language, the ability to change the initiative of the dialogue, or any other facilities which the language designer deems appropriate. For himself, the language designer may include statements which provide dumps or activate traces or timings within his programs.

Through the semantic facilities provided by NDS, a language designer is capable of detecting syntactic or semantic errors in a user's input, informing him of the error, and then allowing him to correct just that part of the text in question. Using this approach a typical interaction might be:

User: PRINT STORE NAME, EARMINGS: WHEN
DEPRECIATION > 40%: GO:
Computer: 'EARMINGS' IS A MISPELLED NAME, REENTER.
User: EARNINGS
Computer: DEPRECIATION CANNOT EXCEED 25%,
REENTER.
User: 20%
⋮

Note that the user need correct only that part of the text which is incorrect.

For certain applications or for certain user experience levels, computer-initiated dialogue is a meaningful style of man-machine inter-

action. A language designer may implement this style of dialogue using the semantic facilities of NDS. The user must initiate the change in the style of the dialogue through a statement in the language. From that point in time, the machine may have the initiative and may interact with the user in a question and answer style illustrated by the following example:

```
   User: HELP:
Computer: WHAT DO YOU WANT TO DO? (PRINT, RANK,
     PLOT)
   User: PRINT
Computer: WHAT INFORMATION DO YOU WANT
     PRINTED?
   User: EARNINGS
Computer: FOR WHICH STORES?
   User: BUFFALO, SYRACUSE
                 ⋮
```

Thus, a wide variety of dialogue styles is obtainable using NDS. The system itself is now described.

IV. AN OVERVIEW OF NDS

As illustrated in Fig. 1, NDS consists of two phases, a setup phase and an execution phase. These two phases interface with two different audiences, a language designer and the set of end-users of the language designer's system.

The language designer prepares a description of his language and dialogue style (details of which will be described later) to be presented to the setup phase of NDS. The setup phase translates these descriptions from a form suitable to the programmer to a form suitable to the execution phase of NDS. These translations are written by the setup phase onto a set of language analysis and dialogue monitor driving files for later access by the execution phase. The language designer also prepares a set of program modules containing programs to perform the tasks corresponding to the verbs and dialogue service statements in the language. At appropriate times during the dialogue, the execution phase of NDS will pass control to these program modules to perform the appropriate tasks.

Users communicate with the system through the execution phase of NDS. The execution phase consists of a dialogue monitor, a language analysis module, and a set of "built-in" dialogue service functions which are accessible to all languages. The dialogue monitor accepts

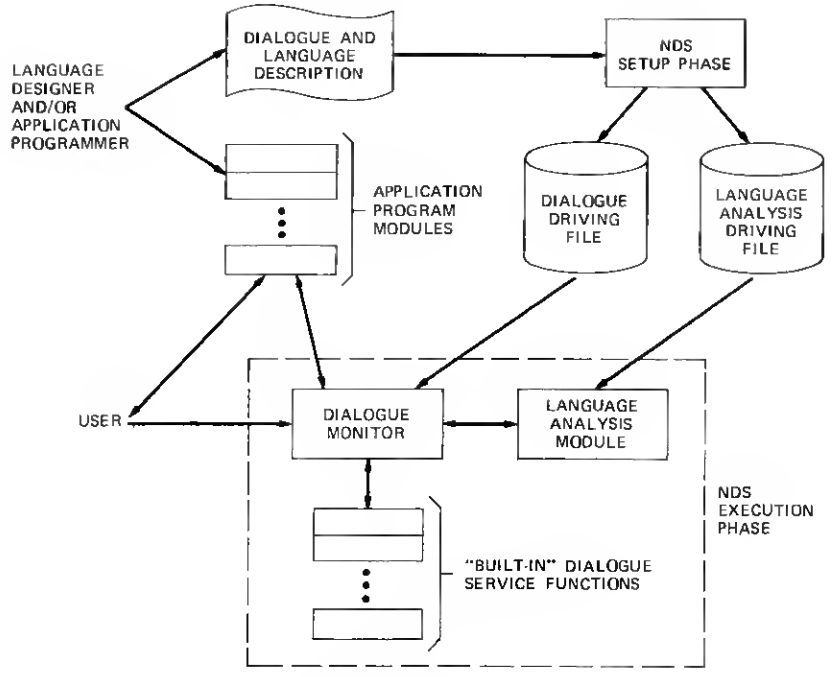

Fig. 1—An overview of NDS.

input from the user of the system in the form of a series of statements, each beginning with a keyword and ending with the character colon (:). The dialogue monitor breaks out the clause of each statement and passes it to the language analysis module together with the clause description as specified by the language designer. The language analysis module attempts to parse and translate the clause into an internal text of integers as specified by the clause description. The algorithm employed by the language analysis module is an extension of the top-down left-to-right algorithm given by Cheatham and Sattley.[6] Successful translations returned by the language analysis module to the dialogue monitor are placed in translation space for later access and a record is kept regarding which statements are currently active in the dialogue.

The GO statement is used to indicate that the user's message to the system is complete. When it is encountered, NDS makes a series of checks, called GO-analysis, which insure that any interstatement relationships declared by the language designer have been fulfilled. There are really two kinds of GO-analysis. One occurs when the current

verb is different from the last verb which was successfully executed. The system checks that a current verb exists, that all required modifiers of the current verb are active, and that no modifier (required or optional) of the current verb is inactive because it was incorrectly given since the last time the GO statement was encountered. The other type of GO-analysis occurs when the current verb is the same as the last verb which was successfully executed. The system makes the same checks described above, but also checks that at least one modifier of the current verb (required or optional) has been correctly given since the verb was last executed. If other interstatement relationships exist, facilities are provided for the language designer to specify additional checks to be executed as part of GO-analysis. If GO-analysis is successful, NDS passes control to the program module corresponding to the current active verb. When execution of the module is complete, control is returned to the dialogue monitor and the dialogue with the user is resumed.

When the dialogue monitor recognizes a dialogue service statement in a user's message, control is passed to the appropriate program module immediately. When execution of the module is complete, control is returned to NDS and the dialogue continues.

NDS provides a set of "built-in" dialogue service statements to provide services common to all languages. These include:

STOP	disconnect the user from the system
RETURN	return control to the host operating system
DELETE	remove rather than replace a currently active statement
CLEAR	remove all currently active statements
RECAP	print out to the user all currently active statements
DETAIL	cause automatic recapping of the current message when the GO statement is recognized
VOCABULARY	print out to the user a list of keywords and their synonyms for all statements in the language
INPUT	direct NDS to take its input from a previously prepared character sequential file
DUMP	print out the translation of a currently active statement or all currently active statements

The language designer may redefine any of these "built-in" dialogue service statements by including in his language a dialogue service

statement of the same name and providing a corresponding program to perform the function he desires. The corresponding "built-in" statement is then not accessible to users of the language.

V. SYSTEM SPECIFICATION

In order to create a system using NDS, the language designer must supply a dialogue description and a language description to the NDS setup phase and prepare a set of programs to be called by the NDS dialogue monitor to perform whatever tasks may be requested by his users. The general forms of these specifications are now described.

5.1 *Dialogue Description*

The dialogue description of a language written using NDS consists of a descriptor for each statement in the language. Each descriptor consists of a series of attributes which are to be assigned to the statement. These attributes define certain properties of the statement and may define certain relationships between the statement being described and other statements in the language. In general, the attributes of a statement are the statement identifier, statement keyword(s), clause syntactic type, translation allocator, verb indicator, required and optional modifier specifications, additional GO-analysis checks, dialogue service indicator, program control information, and Polish indicator.

The statement identifier is a unique positive integer which can be thought of as the internal identifier of the statement. This number is used as a key to locate, store, and interrogate the translation of the statement in translation space. The statement keyword, a unique character string containing no blank characters, is the external identifier of the statement. NDS allows a statement to have an arbitrary number of keyword synonyms which again must be unique for the entire language.

If a statement in a language is to have a clause following its keyword, then the language designer must specify a clause syntactic type as part of the descriptor for that statement. The clause syntactic type is the link between the dialogue descriptor of the statement and that part of the language description which defines the syntax and semantics of its clause. A statement having a clause syntactic type as an attribute must also have a translation allocator. The translation allocator is used to specify the length of the largest possible translation of the clause of the statement. If the user inputs a statement whose trans-

lation size exceeds that indicated by its translation allocator, the dialogue monitor will output a standard message indicating tbat the statement is too long.

If a statement in the language is to be recognized as a verb, tben a verb indicator must be specified as one of the attributes of the statement. A verb statement may require other statements to be present in the current message when the user types tbe GO statement. These are called required modifiers. If a verb requires otber statements to be present, the language designer specifies these statements according to their respective statement identifiers as part of the verb's descriptor. Statements which are not required in the same message witb a verb, but which somehow change the meaning or action of the verb, and may therefore be thought of as optional modifiers, are specified in an identical fashion. If other, more complex relationships are to exist between a verb and other statements in the language, facilities are available for the language designer to include, in the descriptor of the verb, checks of these relationships to be performed as part of GO-analysis.

If a statement is to be recognized as a dialogue service statement, a dialogue service indicator must be part of its descriptor. A dialogue service statement may have a clause, in wbicb case a clause syntactic type and translation allocator must be given as part of its descriptor. For both verbs and dialogue service statements, program control information must be specified as an attribute in the descriptor of tbe statement. This information identifies the program module which contains the program corresponding to tbe verb or dialogue service statement being described.

Tbe Polisb indicator is used to specify tbat tbe clause of a statement consists of a function containing operators and operands and tbat the language designer has followed certain rules in defining tbe clause syntactic type of the statement. When a successful translation of a statement witb the Polish indicator is returned to the dialogue monitor, it will be converted to early-operator Polish postfix notation[7] before being stored in translation space.

5.2 *Language Description*

The clause descriptions for tbe statements in a language are written in a language specification meta-language. This meta-language is really a combination of two distinct languages: a descriptive language which is used to describe tbe syntax or structure of a clause and,

embedded in it, a procedural language, called the Natural Dialogue Semantic Programming Language (NDSPL), which is used to specify context-dependent syntax checks, modifications to the normal clause translations, diagnostic messages, error correction methods, and changes in the initiative of the dialogue.

In the descriptive meta-language, a syntactic type is indicated by a name surrounded by ⟨ and ⟩. A syntactic-type definition consists of a syntactic type followed by an equal sign (read as "is defined as") followed by a sequence of language specification elements which define the syntactic type. The language specification elements are members of the following set:

⟨···⟩	syntactic type
|	exclusive or (alternation indicator)
&	and (conjunction indicator) used to indicate that a portion of a clause is to be parsed and translated in more than one way
[···](*i*, *j*)	a repeating group of specification elements to be repeated at least i times but not more than j times, $j \geqq i \geqq 0$, $j > 0$, $i = j = 1$ if the parenthesized pair is omitted
'···'	a semantic procedure type consisting of one or more NDSPL statements enclosed in primes
"···"(*t*, *p*)	a non-null terminal character string enclosed in quotes, called a literal, with its translation number t and, if the literal is to act as an operator, its precedence p, null translation if the parenthesized pair is omitted
S	any member of the terminal class string, consisting of all non-null character strings
N	any member of the terminal class number, consisting of all numbers, signed or unsigned, with or without a decimal point
V	any member of a language-designer-defined terminal class

The complete specification of a language consists of a syntactic-type definition for each of the clause syntactic types associated with a statement in the dialogue description part of the system description. This specification provides instructions to the language analysis module of the dialogue monitor to parse and translate user input statements. Once the dialogue monitor has recognized a keyword in a user's input statement, it passes the clause following that keyword along with the clause syntactic type associated with the statement to

the language analysis module for parsing and translation. The parser applies the clause syntactic-type definition on the input clause from left to right. If the parser encounters an element representing a terminal class (V, N, S, or literal), it must match an initial substring of the remaining input clause as a member of that terminal class and add to the translation of the clause appropriately. If the parser encounters a syntactic-type definition, it must apply it left to right on the remaining input clause. If the parser encounters a semantic procedure type, it must execute it. When the parser simultaneously encounters the end of the clause syntactic-type definition and the end of the user's input clause, the parse is successful and the completed translation of the clause is returned to the dialogue monitor to be placed in translation space.

Of the language specification elements available to a language designer using NDS, two deserve more detailed discussion: the terminal class V and the semantic procedure type. The terminal class V consists of a set of character strings defined by the language designer. Each member of the class is assigned a set of integer attributes. The occurrence of a V in a syntactic-type definition instructs the parser to match an initial substring of the remaining input clause with a member of this set of character strings, to append the value of its first attribute to the translation, and to make the values of its other attributes available for examination by succeeding semantic procedure types. The use of the terminal class V allows the language designer to specify the skeleton of a language where certain terminal class members must be chosen from a particular set. The composition of this set may then be changed without affecting the language specification.

The semantic procedure type is the means by which the procedural part of the meta-language is embedded within the descriptive part. It consists of one or more NDSPL statements surrounded by primes and can succeed or fail just as all other language specification elements can succeed or fail. The statements available in NDSPL are the following:

SET	arithmetic assignment statement
IF	control for conditional execution (similar to the logical IF statement in FORTRAN IV)
FOR, NEXT	iteration control statements
GOTO	unconditional transfer
STASH	add to the current translation
UNSTASH	remove from the current translation

PRINT	print a message to the user
FAIL	cause unconditional failure of a semantic procedure type
S&F	arithmetic assignment and cause unconditional failure of a semantic procedure type
P&F	print a message to the user and cause unconditional failure of a semantic procedure type
TEST	cause conditional failure of a semantic procedure type
T&P	cause conditional printing of a message to the user and failure of a semantic procedure type
READ	cause a recursive call on the parser to ask the user for additional input and to parse it according to some syntactic-type definition
CALL	cause control to be passed to a language-designer-provided own-code semantic procedure which may succeed or fail

The data which are available for manipulation in NDSPL include numeric constants; a set of language-designer-declared variables; the current translation; the attributes of the most recently matched members of V, N, and S; a set of variables provided by NDS which give a picture of the current state of the parse; and the current state of the dialogue. Messages printed to the user through NDSPL may include any of the above data plus constant character strings; the most recently matched members of V, N, and S; and the character string which the parser most recently attempted to match as a member of V, N, or S.

The semantic procedure type and the facilities of NDSPL give the language designer a powerful tool for creating an interactive language and style of dialogue which are tailored to both his end-user's needs and the needs of his programs. First of all, he has the ability to do context-dependent syntax checks by setting flags or saving the attributes of terminal class members at one point in the parse for later examination to determine what course the parse has taken or should take. He also has the ability to add to, delete from, or modify the normal clause translations using arithmetic functions of the available data. Thus, the translations of the user's input can be tailored to the needs of the application programs. The output facilities of NDSPL provide him with the means to supply his users with timely, relevant error diagnostics when errors are detected in their input statements. Moreover, the READ statement gives him the ability to seize the

initiative in the dialogue. This can he used to ask for error corrections in mid-parse or to change the style of dialogue from user-initiated to computer-initiated.

5.3 *Programs*

Except for any own-code semantic procedures needed by the language, the only programs which must be supplied by the language designer are the programs for each verb and each dialogue service statement in the language. When a user issues a dialogue service statement or executes a verb using the GO statement, the NDS dialogue monitor uses the program control information given in the dialogue description to pass control to the proper program module. The program has access to translation space and to a set of NDS-provided variables which give a picture of the current state of the dialogue. An application program can be as simple or as complex as is necessary to perform the desired task. When execution of the module is complete, control is returned to NDS and the dialogue with the user is resumed.

5.4 *A Simple Illustrative Example*

Suppose that one wishes to implement an information retrieval language which allows users to do scatter plots of one variable versus another. The values for these variables are to come from a data base containing data for the years 1961 to 1973. The plot process is to be implemented as a verb, specifying the variables to be plotted, and one required modifier specifying a range of years for which data values are to be included in the plot. The specified variables must, of course, have numeric values. The user will he allowed to make requests such as

PLOT EMPLOYEES BY REVENUES: FOR 1965–1971: GO:

and

PLOT REVENUES BY EXPENSES: FOR 1961 THRU 1970: GO:

The computer program which has been written to carry out the plot request requires as input the internal numeric identifiers of the two variables and two numbers from one to thirteen, in increasing order, which specify the span of years to be included. The problem is to design a language to translate a user's request into the necessary computer program inputs insuring that a valid request has been made.

Suppose that the terminal class V contains, in part:

Symbol	*Attributes*	
	Varno	*Type*
YEAR	1	1
COMPANY	2	3
EMPLOYEES	3	1
REVENUES	4	2
EXPENSES	5	2

where the symbols are variable names and the attributes of a variable are its internal numeric identifier and its type (1 for integer, 2 for floating point, and 3 for character).

The specification of one possible language for communicating with the plot processor is given in the appendix. This language specification could result in a dialogue similar to the following (user input shown in lower case):

REQUEST⟩ plot employees by revenues: for 1965–1970: go:

The PLOT processor would produce a plot of variables 3 and 4 at level 2 for years 5–10.

REQUEST⟩ plot employees by company:

COMPANY IS A NON–NUMERIC VARIABLE ILLEGAL FOR PLOT

REQUEST⟩ plot revenues by expenses: go:

The PLOT processor would produce a plot of variables 4 and 5 at level 2 for years 5–10.

REQUEST⟩ for 1970 thru 1975:

1975 MUST BE A YEAR BETWEEN 1961 AND 1973 INCLUSIVE

REQUEST⟩ for 1968 thru 1962: go:

The PLOT processor would produce a plot of variables 4 and 5 at level 2 for years 2–8.

REQUEST⟩

⋮

VI. CONCLUSION

NDS provides a means for the easy implementation of time-sharing-based systems which employ a keyword style of man-machine dialogue.

Facilities are available to specify the syntax of the keyword clauses; the forms of their translations; timely, relevant error diagnostics; and a spectrum of dialogue styles, ranging from computer-initiated dialogue to user-initiated dialogue.

NDS has been used to produce a variety of application systems primarily in the area of interactive query languages for information management systems. Use of NDS substantially reduces the programming effort required to implement such systems; and moreover, the implementation may be done utilizing less sophisticated programming talent than would otherwise be necessary.

APPENDIX

```
ATTRIBUTES VARNO, TYPE
SCRATCH CELLS TEMP

NOTVAR = T "IS NOT VARIABLE NAME"
NONNUM = V "IS A NON-NUMERIC VARIABLE ILLEGAL
  FOR PLOT"
BADYR = T "MUST BE A YEAR BETWEEN 1961 AND 1973
  INCLUSIVE"

⟨PLT⟩ = ⟨NUMVAR⟩ "BY" ⟨NUMVAR⟩
*
* IF SYMBOL NOT IN TERMINAL CLASS V, PRINT NOTVAR
* IF SYMBOL IS A NON-NUMERIC VARIABLE, PRINT
*    NONUM
*
⟨NUMVAR⟩ = [V | 'P&F NOTVAR'] 'IF (TYPE = 3) P&F
    NONUM'
*
* IF YEARS NOT GIVEN IN INCREASING ORDER, REVERSE
*    THEM
*
⟨FOR⟩ = ⟨YR⟩ ["THRU" | "−"] ⟨YR⟩
        'IF (TRANS (1) ⟨= TRANS (2)) GOTO X; SET TEMP
        = TRANS (1); SET TRANS (1) = TRANS (2); SET
        TRANS (2) = TEMP; X'
*
* IF YEAR SPECIFIED IS NON-NUMERIC OR IF YEAR OUT
*    OF RANGE,
```

```
* PRINT THE ERROR MESSAGE BADYR
*
* IF VALID YEAR MATCHED AS N, REMOVE IT FROM
*    TRANSLATION AND
* SUBSTITUTE YEAR-1960 WHICH IS A NUMBER BETWEEN
*    1 AND 13
*
⟨YR⟩ = [N | 'P&F BADYR'] 'T&P VAL⟩ = 1960 ·AND· VAL
        ⟨= 1973, BADYR; UNSTASH 1; STASH VAL-1960'

STATEMENT 1: "PLOT": "P": VERB: REQ MOD 2: SYNTAX
  ⟨PLT⟩: MAX TRANS 3: PROGRAM GPLOTR:
STATEMENT 2: "FOR": "F": SYNTAX ⟨FOR⟩: MAX TRANS 2:
```

REFERENCES

1. Sinowitz, N. R., "DATAPLUS—A Language for Real-time Information Retrieval from a Hierarchical Data Base," AFIPS Conf. Proc,. *32*, 1968.
2. Chai, D. T., "Language Considerations for Information Management Systems," unpublished work.
3. Chai, D. T., "An Information Retrieval System Using Keyword Dialog," Inform. Stor. Retr., *9*, July 1973, pp. 373–387.
4. Heindel, L. E., and Roberto, J. T., "The Off-The-Shelf System—A Packaged Information Management System," B.S.T.J., this issue, pp. 1743–1763.
5. Heindel, L. E., and Roberto, J. T., "LANG-PAK—An Interactive Incremental Compiler-Compiler," Proc. ONLINE 72, Int. Conf. Interactive Computing, *1*.
6. Cheatham, T. E., and Sattley, K., "Syntax Directed Compiling," Proc. Eastern Joint Computer Conf. AFIPS, *25*, 1964, pp. 31–57.
7. Hamblin, C. L., "Translation to and from Polish Notation," Comput. J., *5*, October 1962, pp. 210–213.